Multi-Camera Scene Flow by Tracking 3-D Points and Surfels

Frédéric Devernay, Diana Mateus, Matthieu Guilbert

Multi-Camera Scene Flow by Tracking 3-D Points and Surfels

Frédéric Devernay Diana Mateus Matthieu Guilbert
INRIA Rhône-Alpes
38334 Saint Ismier Cedex, France
{frederic.devernay,diana.mateus,matthieu.guilbert}@inria.fr

Abstract

Scene flow represents the 3-D motion of points in the scene, just as optical flow is related to their 2-D motion in the images. As opposed to classical methods which compute scene flow from optical flow, we propose to compute it by tracking 3-D points and surface elements (surfels) in a multi-camera setup (at least two cameras are needed). Two methods are proposed: in the first one, the translation of each 3-D point is found by matching the neighborhoods of its 2-D projections in each camera between two time steps; in the second one, the full pose of a surfel is recovered by matching the image of its projection with a texture template attached to the surfel, and visibility changes caused by occlusion or rotation of surfels are handled. Both methods detect lost or untrackable points and surfels. They were designed for real-time execution and can be used for fast extraction of scene flow from multi-camera sequences.

1. Introduction

Scene flow was introduced by Vedula *et al*. [14, 15] as the 3-D vector field, defined on each point on every surface in the 3-D scene, which represents the motion of these points between two time frames. Since the optical flow is simply a projection of the scene flow onto a camera image plane, the most obvious way to compute scene flow is to reconstruct it from the optical flow measured in one [15] or more cameras [15, 16], possibly helped by a dense stereo reconstruction [9]. In these cases, the difficulty consists in constructing a scene flow which is compatible with several observed optical flows which may bring contradictory informations.

Another approach is to work in the scene domain, and to track 3-D scene points or surface elements (surfels) instead of 2-D image points. The most notable work in this area is perhaps the one of Carceroni and Kutulakos [3]. They model the scene as a set of surfels, each surfel being described by its shape (an oriented planar patch), reflectance (a texture for the albedo and the two specular coefficients of a Phong model), bump map (which models local surface curvature), and motion (modeled as a 3-D affine transform). They show that, given camera parameters and knowing the position of light sources, they can recover the surfel parameter by successive optimizations performed on subsets of the parameters, so that the surfels maximize photo-consistency. The resulting inter-frame 3-D motion field is computed at sampled positions in a known 3-D volume, and the method doesn't try to follow surfels over several frames. The way the method deals with self occlusions is by reconstruction the whole scene so that occlusions can be recovered explicitly. Overall, though the method and the results are impressive, it requires a well-controlled lighting and acquisition setup, and because of the high number of surfel parameters to optimize, it is limited to the recovery of the scene flow in a limited volume. Dellaert *et al*. [4] also propose a surfel tracking method, but their method is more focused on extracting a super-resolved surfel texture, and it is limited to one camera. Pons *et al*. [11] propose a two-step approach, in which they solve alternatively for 3-D reconstruction and scene flow using a variational method. However, their method handles the visibility of each scene point from the cameras using the global 3-D reconstruction, which implies that the whole 3-D scene has to be reconstructed without any missing parts.

We propose two novel methods to compute scene flow which work in the scene domain (as in [3]), but rather than sampling the scene volume and extracting the scene flow at each position in that volume, we propose an approach inspired by classical 2-D tracking, transposed in 3-D. Good tracking candidates are first detected from the original images, the initial surfel pose (*i.e*. translation and rotation) and texture parameters can be reconstructed from at least two images, and then each 3-D point or surfel is tracked over the longest possible time sequence using a multi-camera extension of the Lucas-Kanade algorithm [10, 1]. At each time frame and for each surfel, we extract the translation components from all the images where it is visible using a pyramidal approach, and then we can compute the full pose parameters (rotation and translation). The surfel visibility

is updated between every two time frames, from the surfel orientation with respect to each camera, and from the correlation between the surfel texture and the images where it projects. The result is a set of trajectories across time, where each 3-D point or surfel can become visible or invisible in each camera at each time frame.

Rather than computing a dense scene flow at each time frame as in previous approaches [15, 3], we directly get a set of 3-D trajectories over an interval of time, and scene flow at a given time is obtained by deriving these trajectories. Obviously, further scene motion analysis will be easier to compute from full 3-D trajectories than from instantaneous motion vector fields which need to be integrated over time: trajectories obtained by integrating scene flow may drift from the actual scene motion. In the case of surfel tracking, we make sure that the tracked surfel does not drift from the physical point on the surface by monitoring the intrinsic texture of each surfel.

The rest of the paper is organized as follows: we first describe the general framework for a multi-camera extension of the Lucas-Kanade tracking method, and we then derive it into two particular methods: Tracking of 3-D points, and tracking of 3-D surfels. We then discuss about the initialization of 3-D points and surfels, which is based on the tracking method itself. Finally, we present results of scene flow and trajectories computation on real scenes and present conclusions and further work on these methods.

2. Multi-camera extension of Lucas-Kanade

The Lucas-Kanade method is a classical method to compute optical flow or to track 2-D points in a video sequence, and a reference publication on this subject is the study by Baker & Matthews [1], from whom we borrowed most of the notation used in this paper. They give four formulations of this problem (forward or backward, additive of compositional), from which we use the *forward additive* method, for reasons which will be explained in the conclusion.

In the Lucas-Kanade problem, each tracked feature is described by a vector of parameters $\mathbf{p}$ and a texture template $T(\mathbf{x})$, where $\mathbf{x}$ is a texture point. Given feature parameters $\mathbf{p}$, the *warp* function $\mathbf{W}(\mathbf{x};\mathbf{p})$ maps each point $\mathbf{x}$ in the texture template T to a point in the image I. The method optimizes the feature parameters in order to minimize an energy formed by the squared differences between the texture template and the image:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right]^2 . \tag{1}$$

In our case, since there are several cameras, the appearance of the feature may be different in each camera n, consequently there may be different texture templates T_n attached to each warp $\mathbf{W}_n$. A feature may be more or less visible in each camera, which can be expressed by a positive weight v_n, which we call *visibility*. The energy becomes:

$$\sum_n v_n \sum_{\mathbf{x}} \left[I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p})) - T_n(\mathbf{x})\right]^2 . \tag{2}$$

The Lucas-Kanade algorithm supposes that an estimate of $\mathbf{p}$ is known, so that, at each optimization step, the goal is to find $\Delta\mathbf{p}$ which (approximately) minimizes:

$$\sum_n v_n \sum_{\mathbf{x}} \left[I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T_n(\mathbf{x})\right]^2 \tag{3}$$

Using the first order Taylor expansion of I_n in Eq. (2) to expand Eq.(3) gives:

$$\sum_n v_n \sum_{\mathbf{x}} \left[I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p})) + \nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}} \Delta\mathbf{p} - T_n(\mathbf{x})\right]^2 , \tag{4}$$

and its partial derivative with respect to $\Delta\mathbf{p}$ is:

$$2\sum_n v_n \sum_{\mathbf{x}} \left[\nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}}\right]^{\mathsf{T}} \left[I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p})) + \nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}} \Delta\mathbf{p} - T_n(\mathbf{x})\right] . \tag{5}$$

At the minimum, this partial derivative must be zero, which leads to the following expression for the parameters update:

$$\Delta\mathbf{p} = H^{-1} \sum_n v_n \sum_{\mathbf{x}} \left[\nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}}\right]^{\mathsf{T}} \left[T_n(\mathbf{x}) - I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p}))\right] , \tag{6}$$

where H is the Gauss-Newton approximation of the Hessian matrix:

$$H = \sum_n v_n \sum_{\mathbf{x}} \left[\nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}}\right]^{\mathsf{T}} \left[\nabla I_n \frac{\partial \mathbf{W}_n}{\partial \mathbf{p}}\right] . \tag{7}$$

The parameters are then updated ($\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$), and the procedure is iterated until convergence (usually given by $\|\Delta\mathbf{p}\| < \epsilon$).

The next two sections specialize this multi-camera extension of Lucas-Kanade for tracking 3-D points and 3-D surfels using several cameras.

3. Tracking 3-D points using several cameras

3.1. Lucas-Kanade Iteration

When the tracked features are 3-D points, the parameters vector is simply made out of the world coordinates of this point, $\mathbf{p} = (X, Y, Z)$. In order to compute the parameters update (6), we need to define the following ingredients: The

texture templates $T_n(\mathbf{x})$, the warps $\mathbf{W}_n$, and the Jacobian of each warp $\frac{\partial \mathbf{W}_n}{\partial \mathbf{p}}$.

The templates $T_n(\mathbf{x})$ are square windows extracted from the set of previous images centered around the sub-pixel projection in each image of the 3-D position at the previous time frame (bilinear interpolation is used for re-sampling the images).

Recall that the warps $\mathbf{W}_n$ are 2-D functionals that map template coordinates to image coordinates in image n. In this case, each warp $\mathbf{W}_n$ is the translation in image n by the 2-D coordinates of the projection of $\mathbf{p} = (X, Y, Z)$ in that image (we suppose in the rest of this paper that the projection is perspective and that nonlinear distortion was removed from the images):

$$\mathbf{W}_n(\mathbf{x}; \mathbf{p}) = \mathbf{x} + \mathbf{P}_n(\mathbf{p}), \tag{8}$$

where the coordinates (x, y) of the projection can be written from the projection matrix $\tilde{\mathbf{P}}_n = (p_{ij})_{3\times 4}$ of camera n using homogeneous coordinates,

$$(sx, sy, s) = \tilde{\mathbf{P}}_n \tilde{\mathbf{p}}, \text{ with } \tilde{\mathbf{p}} = (X, Y, Z, 1). \tag{9}$$

If we write the projection function $\mathbf{P}_n$ without using homogeneous coordinates, we can compute the Jacobian of the warp in image n:

$$\frac{\partial \mathbf{W}_n}{\partial \mathbf{p}} = \frac{1}{s} \begin{pmatrix} p_{11} - x p_{31} & p_{12} - x p_{32} & p_{13} - x p_{33} \\ p_{21} - y p_{31} & p_{22} - y p_{32} & p_{23} - y p_{33} \end{pmatrix}, \tag{10}$$

and the parameters update (6) is then computed from the Hessian (7). Of course, given the expression of the Jacobian, the Hessian H will be of rank 2 (thus non-invertible) if there is only one camera in which the point is visible (*i.e.* $\exists n_0, v_{n_0} > 0$ and $\forall n \neq n_0, v_n = 0$); this is reasonable, since the 3-D position of a point cannot be recovered from only one camera.

3.2. Computing the visibility

Usually, the appearance of a 3-D point changes slowly, but its visibility may change abruptly, especially when it becomes occluded by another part of the scene. If only two cameras are used, each point has to be seen in both cameras, so we can only assume $\forall n, v_n = 1$.

If the point is visible in 3 cameras or more, the energy (2) can be considered as a weighted least-squares problem with outliers, where the visibility values v_n are weights which can be obtained using a robust estimator such as Huber's M-estimator [7]. First of all, one must estimate the robust scale σ of our least squares problem, *i.e.* the standard deviation of the residuals if outliers were discarded, supposing the remaining residuals follow a normal distribution. This can be done using the *MAD* (median absolute deviation) scale estimator: $\sigma = 1.4826 \operatorname{med}_n\{|Y_n - \operatorname{med}_m Y_m|\}$, where med denotes the median, and the Y_n are the signed residuals. However, in our situation, the residuals in the energy (2) are grouped per image, and the resulting grouped residuals are positive, so that we cannot use the above expression. We therefore have to simplify the *MAD* scale estimator to the following expression:

$$\sigma = 1.4826 \operatorname*{med}_n \{|Y_n|\}, \tag{11}$$

$$\text{with } Y_n = \sqrt{\sum_{\mathbf{x}} \left[I_n(\mathbf{W}_n(\mathbf{x}; \mathbf{p})) - T_n(\mathbf{x})\right]^2}. \tag{12}$$

Once we have the scale, the visibility values can be estimated using the Huber function:

$$v_n = 1 \text{ if } |Y_n| \leq \sigma\tau, \text{ and } v_n = \frac{\sigma\tau}{|Y_n|} \text{ if } |Y_n| > \sigma\tau, \tag{13}$$

where the 95% asymptotic efficiency on the standard normal distribution is obtained with the tuning constant $\tau = 1.345$. Note that this assures $v_n = 1$ for at least 2 cameras.

These visibility values are re-estimated at the beginning of each iteration, before computing the Hessian (7) and the parameters update (6), leading to a 3-D point tracker which is more robust to visibility changes and occlusions. Using a robust estimator in the case where the number of measurements is at most equal to the number of cameras may not be fully justified, however it still seems to be an appropriate way to handle automatically these visibility changes when little is known about the scene geometry, or when the scene surface is not smooth enough to use surfel tracking, as shown by the experiments.

3.3. Dropping lost points

If, during an iteration, the condition number of H (*i.e.* the ratio of the smallest singular value to the largest singular value) is very small, the parameters update will probably be wrong, so we consider the point is not trackable anymore and deactivate it.

The main problem with the 3-D point tracker is that nothing ensures that the tracked point remains on the surface scene: there can be a drift coming from small 3-D errors accumulated over time. One way to prevent this is to setup two of the cameras as a stereo pair with a small-enough baseline (we will see in section 5 that this is also useful for 3-D point initialization): if, after complete optimization, a 3-D point is visible (*e.g.* $v_n > 0.8$) in both cameras, then the cross correlation between windows centered at each projection in these cameras should be above some threshold (0.8 in our experiments), else the 3-D point is considered lost and the track is cut before the current frame.

3.4. Pyramidal implementation

The main limitation of the Lucas-Kanade method is that, depending on the window size and the texture template, it

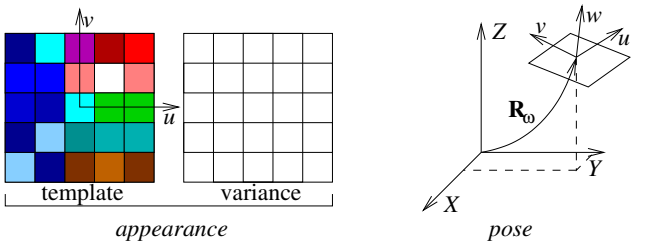

Figure 1. A surfel is defined by its *appearance* (a texture template and its variance) and *pose* (a position and rotation in 3-D)

may only be capable of tracking motions in the images of about one pixel between two time frames. In order to track larger motion, a pyramidal implementation based on the 2-D tracker by Bouguet [2] is employed, which first estimates the 3-D motion at a coarse resolution (typically $1/8$ of the original image size), and then improves it at each finer scale ($1/4$, $1/2$, and 1). When switching from one scale to the other, the parameters vector is unchanged, but the projection matrices must be appropriately resized for the corresponding image resolution.

4. Tracking 3-D surfels

4.1. Definitions

A 3-D surfel is defined as a small planar square region in 3-D space with a *pose* and an *appearance* (Fig. 1).

The *appearance* can be represented as a texture $T(\mathbf{x}), \mathbf{x} \in [-s, s] \times [-s, s]$, and the resolution (r_u, r_v) of this texture which gives the size of a texture pixel in world units. The resolution is chosen so that the projected surfel resolution is about the same as the resolution of the images. Each surfel has a reference frame (u, v, w) attached to it, where u and v are aligned with the texture image axes, and w is aligned with the normal to the surfel. The appearance of a surfel can be extracted at the first time frame it appears, using the inverse of the warp $\mathbf{W}_n$ in each camera where it is visible. However, since the warp involves a perspective projection, the first time frame may not be the best one to get the best appearance. For that reason, the texture template is updated at each time frame, as we will see Sec. 4.4, and the estimated variance of the intensity at each texture pixel $P(\mathbf{x})$ is kept together with the texture template $T(\mathbf{x})$.

The *pose* of a surfel has 6 degrees of freedom, and can be represented by the position (X, Y, Z) of the center of the surfel, and three parameters for the rotation from the world reference frame to the surfel reference frame (we consider only surfels with a rigid motion, although this could be extended to more complicated local deformations such as an affine motion). Since the function that maps the rotation parameters to the actual rotation should not have singularities, we use a rotation vector $\omega = (\omega_X, \omega_Y, \omega_Z)$ to parameterize the rotation, and the rotation matrix can be computed using Rodrigues' formula.

Since the appearance of a surfel usually changes slowly across time, it is updated in a separate procedure, and the parameters vector only contains the pose parameters:

$$\mathbf{p} = (X, Y, Z, \omega_X, \omega_Y, \omega_Z). \tag{14}$$

4.2. Surfel tracking

In the energy (2), we considered the general case were there could be one texture template per camera. In the case of surfel tracking, there is only one texture template T for all the cameras: $T_n(\mathbf{x}) = T(\mathbf{x})$. The warp from the surfel texture template to camera n is a homography caused by perspective projection, composed by the following successive transforms: scaling up texture pixel units to world units, a translation and rotation corresponding to the surfel pose, and finally a projection in camera n. Let $\mathbf{W}_n(u, v)$ be the warp in Euclidean coordinates, and let $\tilde{\mathbf{W}}_n$ be the warp that applies to the homogeneous texture coordinates vector $(u, v, 1)$. It can be represented by the following 3×3 matrix:

$$\tilde{\mathbf{W}}_n = \tilde{\mathbf{P}}_n \begin{pmatrix} r_{11} & r_{12} & X \\ r_{21} & r_{21} & Y \\ r_{31} & r_{32} & Z \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_u & 0 & 0 \\ 0 & r_v & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{15}$$

where the middle matrix is formed from the two first columns of the rotation matrix $\mathbf{R}_\omega = (r_{ij})$ and the translation vector (X, Y, Z) corresponding to the surfel pose.

Let $(a\, b\, c)^\mathsf{T} = \tilde{\mathbf{W}}_n (u\, v\, 1)^\mathbf{T}$, the expression for the Jacobian of the texture warp can simply be obtained by differentiation of the warp $\mathbf{W}_n(u, v) = (a/c, b/c)$ with respect to the surfel parameters $\mathbf{p}$ (the complete expression for the Jacobian is too long to be included here).

Again, if the condition number of H (the ratio of its smallest singular value to its largest singular value) is very small (*e.g.* less than 10^{-8}), we consider the surfel untrackable and deactivate it. Note that whereas 3-D point tracking needs at least two cameras, surfel tracking can work with only one camera, as described by Dellaert *et al.* [4], but some surfel configurations are degenerate (*i.e.* $\det H = 0$), in particular when the surfel plane passes through the optical center.

4.3. Computing the visibility

We can use the same method as with 3-D point tracking to compute the visibility of the surfel in each camera, *i.e.* apply robust estimation to the least squares problem of Eq. 2. However, in the case of surfel tracking, we can also use a more geometric approach, by using the surface of the warped surfel as the visibility. One method to compute this surface is to use an approximation of the projective warp by the tangent affine warp at the surfel center (this approximation is valid as long as the surfel projects to a small area

in the camera image). If the projective warp in homogeneous coordinates (15) is written $\tilde{\mathbf{W}}_n = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$, then the tangent affine warp at $(0,0)$ is given by a first order Taylor expansion of $\mathbf{W}_n$:

$$\mathbf{W}_n^{\text{affine}} = \begin{pmatrix} \frac{ai-cg}{i^2} & \frac{bi-ch}{i^2} & \frac{c}{i} \\ \frac{di-fg}{i^2} & \frac{ei-fh}{i^2} & \frac{f}{i} \end{pmatrix}, \tag{16}$$

and the surface of the projected surfel is proportional to the determinant of the left 2×2 sub-matrix of $\mathbf{W}'_n$, or zero if this determinant is negative:

$$v_n = \max\left(0, \det\begin{pmatrix} \frac{ai-cg}{i^2} & \frac{bi-ch}{i^2} \\ \frac{di-fg}{i^2} & \frac{ei-fh}{i^2} \end{pmatrix}\right). \tag{17}$$

Nevertheless, this geometric approach cannot handle surfel occlusion by other parts of the scene. To detect occlusions, we compute the cross-correlation between the surfel texture template and the warped image in each camera. If the result is under some threshold (*e.g.* 0.8), the surfel's visibility in this camera is set to zero. Of course, occluded surfels can only be detected after pose optimization, so this step has to take place after pose estimation at each time frame. The geometric visibility computation described above could be done at each iteration of the optimization, but since the surfel orientation usually moves slowly between two time frames it can also be done once the pose estimation is complete.

4.4. Texture template update

The texture template is extracted at the first time frame from the camera where it is the most visible, using the inverse of the warp in that image. Since we cannot perform occlusion detection without first having the texture template, it is better to extract it from one of the camera images used for surfel initialization (Sec. 5). Since the images from the first time frame contain intrinsic noise, and the pose and visibility of the surfel vary over the sequence, the texture extracted at other time frames may bring more information about the texture template, and we can incorporate this information by updating the texture template $T(\mathbf{x})$ and its variance $P(\mathbf{x})$.

We propose to use a discrete Kalman filter (DKF) for that purpose, which is a simplified version of what was proposed by Dellaert *et al*. [4] to build a super-resolved texture map. Basically, the DKF needs several ingredients: A state equation, a measurement equation, the state parameters covariance, and the measurement noise covariance. The state equation is trivial, since the state is the texture template, and it is supposed to be constant over time. The measurement equation is trivial too: we measure the state itself, by inverse warping of each image where the surfel is visible, and the measurement noise covariance is supposed to be diagonal (which is disputable, since neighboring pixels will most certainly have correlated values after applying the inverse warp, because of re-sampling). The DKF update equations can then be written for each camera n and at each template pixel $\mathbf{x}$ independently:

$$K = P(\mathbf{x})/(P(\mathbf{x}) + V_n(\mathbf{x})) \tag{18}$$

$$T(\mathbf{x}) \Leftarrow T(\mathbf{x}) + K\left(I_n(\mathbf{W}_n(\mathbf{x};\mathbf{p})) - T_n(\mathbf{x})\right) \tag{19}$$

$$P(\mathbf{x}) \Leftarrow P(\mathbf{x})(1-K), \tag{20}$$

where $V_n(\mathbf{x})$ is the measurement variance, K is the Kalman gain, and $P(\mathbf{x})$ is the texture (and state) variance. The state variance and measurement variances should be expressed in squared intensities, but using an arbitrary multiple of the intensity will not change the update equations: multiplying $P(\mathbf{x})$ and V_n in the equations above does not change the value of $T(\mathbf{x})$. Therefore, we can express V_n in any unit, $P(\mathbf{x})$ will be in the same unit. Intuitively, the measurement noise should decrease when the visibility increases, and $\lim_{v_n \to 0^+} V_n = +\infty$, which led us to an empirical expression for measurement noise variance: $V_n = \frac{1}{v_n^2}$. The DKF equations are used after pose and visibility update, to update every template pixel using all cameras where it is visible.

Dropping lost surfels. Wrong surfels can be detected from the condition number of H (Sec 3.3), or when the visibility of a surfel is zero in all cameras. In these cases, the surfel is considered lost and the track is cut before the current frame.

Handling large motion. Of course, surfel tracking suffers from the same problems as 3-D point tracking when large motion occur in the images. However, since surfel tracking needs very precise intensity measurements to get the surfel orientation, a pyramidal implementation (Sec. 3.4) will probably give wrong results. We propose instead to bootstrap the surfel tracking at each time frame using the 3-D point tracking (Sec. 3). This update the surfel position (X, Y, Z), and all six pose parameters are then estimated using surfel tracking. This procedure gave satisfying results in our experiments.

5. Good 3-D points or surfels to track

In the classical 2-D point tracking algorithms, the warp function $\mathbf{W}$ is usually a simple image translation, and points are selected in the first image by comparing the eigenvalues of matrix H (7) at every image point for that family of warps, and keeping the best candidates [6, 12]. More recent work extended this to any warp in the image or intensity space [13], but the case of 3-D tracking has not been studied yet. Of course, the complexity is very different, since we would have to compute the eigenvalues of H

for every point in parameter space, which means every point in 3-D space for point tracking, and every point in 6-D space for surfel tracking.

Initializing 3-D points. We propose another approach to select points or surfels: let us suppose that some 3-D points can be matched between the images from the first time frame (two of our cameras in the experimental setup have a small baseline for that purpose). Since both 3-D point tracking and surfel tracking start with a 3-D point tracking step, we can compute the Hessian (7) from the Jacobian of the 3-D warp (10) at every matched point. We can then select those for which the smallest eigenvalue and/or the condition number of H are above some threshold. Since the point visibility is not available at this state of the process, only the images used to reconstruct the 3-D point can be considered.

If we use a stereo pair to reconstruct the 3-D points, each point is considered visible only in these two images. Since the neighborhood of the matched points is usually similar due to the stereo matching algorithm, the selected points are in fact close to the points found by a classical Harris-Stephens corner detector [6]. However, in general, this method can also be used to select 3-D points reconstructed by any reconstruction algorithm, including wide-baseline and multi-camera stereo, and give results that are very different from 2-D feature detectors.

Initializing surfels. If we intend to do 6-D tracking, we also need the local normal to the scene surface, which can be estimated using a method similar to Lucas-Kanade tracking with an affine warp [5]. The surfel texture template can then be initialized from the image used for its reconstruction where it is most visible, and its texture template variance can be set to a large value. We can then update its visibility (Sec. 4.3) and run the surfel tracking method from the first time frame to the first time frame, in order to refine its position and update its texture template. The surfel is then ready to be tracked over the whole sequence.

6. Results

We present some experimental results obtained on a 5-camera system mounted on a desktop (Fig. 2). The cameras are 640×480 B&W cameras with 6mm lenses, calibrated with a multi-camera calibration method. Two of the cameras have a small baseline and are used for 3-D points and surfel initialization by using a dense stereo reconstruction method. The other cameras are used during tracking only, which explains why the tracked points and surfels in these examples are on parts of the scene that were facing the stereo cameras at the first time frame. This is only due to the initialization method we used in these experiments, and is not a restriction of the tracking methods presented here. No points or surfels are added during the sequences, and the points that are still there at the end of the sequence have been tracked since the very first frame.

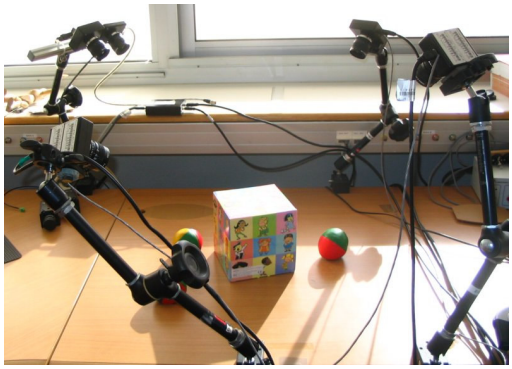

Figure 2. The 5-camera setup used for the experiments. Two of the cameras (top-left) are used as a stereo pair for initialization.

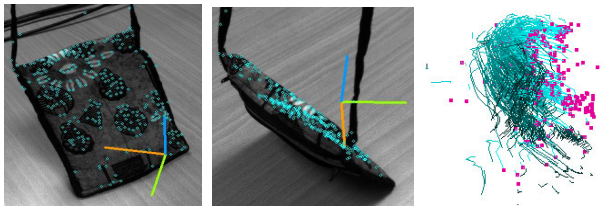

Figure 3. 3-D point tracking results (sequence 2): 300×300 sub-images from cameras 1 (used for initialization) and 5 (showing a visibility problem), and 3-D trajectories.

Sample tracking results are shown Fig. 3 and 4, but most of the results are presented in the additional material that goes with this paper. Several 5-camera sequences are shown for each tracking method (4 sequences for 3-D point tracking, 6 sequences for surfel tracking), and most sequences were tested with both methods. The results are presented both on the original image sequences, and as 3-D trajectories which permits a better (though not perfect) visualization of the scene flow. In the 3-D point tracking sequences, all 3-D points are drawn on all camera images, since the visibility v_n is always strictly positive. In the surfel tracking sequences, the full surfel is drawn in a camera image if it is visible (*i.e.* $v_n > 0$), whereas only the center-point is drawn if the surfel is not visible. The main observation is that whenever surfel tracking fails (*i.e.* lots of surfels are lost), 3-D point tracking still gives good results. However, when the scene surface is smooth and textured (such as in sequence 1, the "drawing" sequence), surfel tracking gives much better results.

7. Discussion and perspectives

7.1. Which method: 3-D points or surfels?

We presented a multi-camera extension of the Lucas-Kanade algorithm, from which we derived two feature

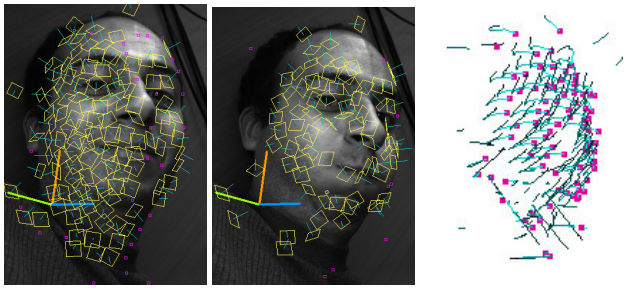

Figure 4. Surfel tracking results (sequence 3): 350×480 sub-images from camera 1 at time frame 0 and 50, and 3-D trajectories. Invisible surfels are marked by a small square, and visible surfels are drawn in actual size with their normal.

trackers: one that tracks 3-D points parameterized by their world coordinates (X, Y, Z), and one that track planar surface elements (surfels) parameterized by their full 6-degrees-of-freedom pose $(X, Y, Z, \omega_X, \omega_Y, \omega_Z)$. These two methods are used to recover scene flow and trajectories over a period of time. Of course, the latter method brings more information on the scene flow and handles visibility in a more rigorous way, but it still suffers from several weaknesses.

The first problem comes with surfel initialization: the scene surface has to be both locally planar *and* sufficiently textured in order to be able to extract the surface normal from the initial images. The surfel tracking method also needs enough local texture to be able to follow the surface orientation across time. Overall, these qualities are difficult to find in a natural scene, though they might be met by some parts of the scene (cloth, printed material, textured surfaces such as wood...). Besides, surfel tracking is more computationally expensive, because of the complexity of the warps involved in this method, and it could be difficult to track several hundred surfels in real-time.

On the opposite, 3-D point tracking is easy to initialize with any multi-camera reconstruction method, and its computational cost is roughly n times the cost of tracking 2-D points in a single view, where n is the number of cameras. It can easily benefit from highly optimized and robust implementations of the standard 2-D Lucas-Kanade point tracking method [2], and runs in real-time on our 5-camera setup where computations are made on a single 2.8GHz Pentium 4. However, automatic visibility handling by the use of M-estimators involves a bit of magic, because we are using a robust estimation on a little number of measurements, and it sometimes fails. The method may also suffer from drift problems, where small errors on the 3-D position estimation are accumulated over time, and the 3-D point ends up not being on the surface scene anymore.

A good balance would be to use surfel tracking only at points in the scene where it will give robust results, *i.e.* on smooth textured surfaces, and to use the 3-D point tracker everywhere else. In fact, if an initial 3-D reconstruction of the scene is available, the 3-D point tracker can be used to recover *scene flow* at almost every point (*i.e.* except where the condition number of H is too small, see Sec. 3.3).

7.2. Why the forward additive method?

As noted in Sec. 2, among the four possible formulations of the tracking problem [1], we picked up the *forward additive* method, which is the original formulation by Lucas and Kanade [10] and the most straightforward, in which the we optimize a full warp from template space to image space. The other formulations are equivalent, but usually have a lower computational cost. For example, the *forward compositional* formulation iteratively solves for an incremental warp, and would require the energy (2) to be rewritten as:

$$\sum_n v_n \sum_{\mathbf{x}} \left[I_n(\mathbf{W}_n(\mathbf{W}_n(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T_n(\mathbf{x}) \right]^2 . \quad (21)$$

The gain in terms of computational cost is due to the fact that the Jacobian of the warp only has to be computed at $\mathbf{p} = 0$ and can be precomputed once for all (the *inverse compositional* formulation also enables pre-computation of the Hessian (7)). However, it requires that the set of warps contains the identity ($\mathbf{W}_n(\mathbf{x}; 0)$ has to be the identity warp), and this condition does not hold in the formulation we use for surfel tracking: the warp could be the identity in *one* camera, but because of perspective projection of the texture template it cannot be the identity in *all* cameras for one value of $\mathbf{p}$. For this single reason, both the *forward compositional* and *inverse compositional* formulations cannot be used for surfel tracking. The *inverse additive* formulation imposes even further constraints on the set of warps and cannot be used for surfel tracking either.

For the 3-D point tracker, it is possible to use the *compositional* formulation by reworking the warp parameterization so that it contains the identity warp. Nevertheless, the computational cost reduction with respect to the *forward additive* formulation would not be significant: the warp in each image consists only in translations, so that both the Jacobian and the Hessian computations are inexpensive. For the sake of simplicity, we preferred using the *forward additive* formulation for both methods.

7.3. Perspectives

Although scene flow was introduced seven years ago by Vedula *et al.* [14, 15], it still has not gained much attention from the community, and a lot of work is still concentrated on either doing static 3-D reconstructions from multiple cameras, or extracting 2-D optical flow from monocular image sequences. Scene flow is at the crossing of these two techniques, and it should be considered as an essential tool to study motion in 3-D scenes, especially articulated

and deformable motion. The main problem probably lies in the fact that reconstructing scene flow is still a difficult process: Vedula *et al*. proposed to reconstruct it from optical flow, but optical flow computation is already an ill-posed problem, whereas other attempts (most notably the work of Carceroni and Kutulakos [3] and Pons *et al*. [11]) involved a complicated optimization framework.

Several problems make scene flow estimation more difficult than optical flow. There is a representation problem: the objects to track are not pixels in the image but small 3-D primitives that have a small footprint in the images. But there is also a visibility problem: these primitives may go from one camera to the other, may become occluded by other parts of the scene, or may disappear completely.

The methods we propose to compute scene flow are based on the much-studied Lucas-Kanade algorithm and its derivatives. Its extension to multiple cameras lead us to two tracking methods: one is capable of tracking 3-D points with automatic handling of the visibility based on robust estimation, and the other tracks surfels (surface elements) and computes the visibility from the geometry of the surfel. They rely on standard least-squares optimization, which can easily be extended to use more robust tracking techniques such as particle filtering. Scene flow can be extracted from as little as two cameras, but any number of cameras can be used without any restriction on their field-of-view, and a complete 3-D reconstruction of the scene is not necessary.

These methods already give satisfactory results, but they still need to go through a validation against ground-truth measurements. Once validated, they will probably represent a very simple and computationally affordable way of computing scene flow data.

References

[1] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *IJCV*, 56(3):221–255, Feb. 2004.

[2] J.-Y. Bouguet. Pyramidal implementation of the Lucas-Kanade feature tracker. Technical report, Intel Corp., Microprocessor Research Labs, 2000.

[3] R. L. Carceroni and K. N. Kutulakos. Multi-view scene capture by surfel sampling: From video streams to non-rigid 3D motion, shape and reflectance. *IJCV*, 49(2-3):175–214, 2002.

[4] F. Dellaert, C. Thorpe, and S. Thrun. Super-resolved texture tracking of planar surface patches. In *IEEE/RSJ Int'l Conf. on Intelligent Robotic Systems*, Oct. 1998.

[5] F. Devernay and O. Faugeras. Computing differential properties of 3-D shapes from stereoscopic images without 3-D models. In *Proc. CVPR* [8], pages 208–213.

[6] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conf.*, pages 189–192, 1988.

[7] P. Huber. *Robust Statistics*. John Wiley & Sons, New York, 1981.

[8] IEEE Comp.Soc. Seattle, WA, June 1994.

[9] R. Li and S. Sclaroff. Multi-scale 3D scene flow from binocular stereo sequences. In *WACV/MOTION*, pages 147–153, Breckenridge, CO, USA, 2005. IEEE Comp.Soc.

[10] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[11] J.-P. Pons, R. Keriven, and O. Faugeras. Modelling dynamic scenes by registering multi-view image sequences. In *Proc. CVPR*, San Diego, CA, June 2005. IEEE Comp.Soc.

[12] J. Shi and C. Tomasi. Good features to track. In *Proc. CVPR* [8], pages 593–600.

[13] B. Triggs. Detecting keypoints with stable position, orientation and scale under illumination changes. In *Proc. 8th ECCV*, pages 100–113, Vol. IV, Prague, Czech Republic, May 2004.

[14] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *Proc. 7th ICCV*, pages 722–729, Kerkyra, Greece, 1999. IEEE Comp.Soc., IEEE Comp.Soc. Press.

[15] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. *IEEE*, 27(3):475–480, 2005.

[16] Y. Zhang and C. Kambhamettu. Integrated 3D scene flow and structure recovery from multiview image sequences. In *Proc. CVPR*, pages 2674–2681, Hilton Head, SC, USA, June 2000. IEEE Comp.Soc.